**CMSC 28400 Introduction to Cryptography** **Autumn 2021**

# Notes #8: Randomized Encryption and Chosen-Plaintext Attack Security

*Instructor: David Cash*

So far we've studied stream ciphers and block ciphers. We previously discussed the pseudo-OTP, and how it can be built with any stream cipher. In future notes we'll look at how to build ciphers for large messages from block ciphers as well.

For these notes we consider the goal of securely encrypting several messages, towards addressing the one-time weakness of the OTP and Pseudo-OTP.

## 8.1 Randomized Encryption

The OTP and psuedo-OTP are *ciphers*, which means they are functions satisfying a certain property we defined in the very first set of notes. Implicit in the definition of a cipher is that encryption is "deterministic," meaning that if one encrypts the same message with the same key multiple times, then the same ciphertext will be emitted each time. A closer look at these ciphers reveals that their security behavior when encrypting multiple messages is even more concerning. For instance, suppose we use the OTP to encrypt a message $m_1$, producing a ciphertext $c_1$. Later, say we want to encrypt message $m_2$ that is exactly $m_1$, but with its first bit flipped. Using the same key, we would produce a ciphertext $c_2$ that is equal to $c_1$, except with its first bit flipped!

Should this property concern us, as cryptographers? Security always depends on the application, and the OTP has its place. But here is an example of the sort of thinking that suggests we should consider stronger security: Suppose we encrypt a super-secret number $m$ using the OTP, which is the number of widgets sold so far this year. Each day we add to $m$ the number of sales, and encrypt the number again, producing a new ciphertext. Now consider the point of view of an adversary: It will see a sequence of ciphertexts, and be able to detect which bits have been changed each time. This will tell the adversary a lot about how our day-to-day sales, even though everything is encrypted, and each ciphertext is (in some sense) "unbreakable" on its own.

This issue played a role in some classical ciphers, like the homophonic cipher you broke in Project 1. Recall that this wasn't really a cipher, since the encryption process was not really a function but a recipe of sorts which could produce different ciphertexts when run on the same input twice. Another example is the famous Enigma cipher used in the mid 20th century, notably by the Germans in World War II. That cipher explicitly instructed senders to pick a random "message key" to *randomize* ciphertexts.

Basically all commonly-used modern encryption is designed to avoid these problems when encrypting multiple messages with the same key. (The ones that aren't are explicitly marked as good for one-time use only.) Because of the deterministic nature of a cipher $E$, we need a new definition that allows for randomization. The following allows for this, and is called an *encryption scheme.* The name is just a convention, but it is pretty standard. Intuitively, a randomized encryption scheme is like a cipher, except the encryption direction takes an extra input that can be used to randomized the process. Decryption still have to recover the message, but doesn't to need recover that extra input.

**Definition 8.1.** *A pair of functions* $\Pi = (\mathsf{Enc}, \mathsf{Dec})$,

$$\mathsf{Enc} : \mathcal{K} \times \mathcal{M} \times \mathcal{R} \to \mathcal{C}$$

*and*

$$\mathsf{Dec} : \mathcal{K} \times \mathcal{C} \to \mathcal{M},$$

*is called a* randomized encryption scheme with key-space $\mathcal{K}$, message-space $\mathcal{M}$, randomness-space $\mathcal{R}$, and ciphertext-space $\mathcal{C}$ *if for every* $k \in \mathcal{K}$*,* $m \in \mathcal{M}$*,* $r \in \mathcal{R}$*, and* $c \in \mathcal{C}$

$$\mathsf{Enc}(k, m, r) = c \quad \Longrightarrow \quad \mathsf{Dec}(k, c) = m.$$

The intended usage is for someone to pick $r \in \mathcal{R}$ and then run $c \leftarrow \mathsf{Enc}(k, m, r)$. An interesting point is that decryption must work with $k$ and $c$ *only.* In particular, the value $r$ chosen by the sender is not assumed to be available for decryption unless it is communicated in $c$; Of course one can always put $r$ there, but doing so may have implications for security.

The OTP and pseudo-OTP do not really fit this definition (if $\mathcal{R}$ is a trivial set or something they can be formally viewed this way, but that is not the spirit of the definition). Intuitively, it's not even clear syntactically how to mix in randomness into these ciphers. In the next set of notes we'll begin looking at how randomness might be used effectively, but with a block cipher. For these notes will define a new security notion for randomized encryption and give a few examples.

## 8.2 Analyzing the Security of a Cipher

Since about the 1980's, an approach for thinking about security has been developed that extends Shannon's idea of giving *mathematical definitions* of security. For better or worse, this approach is called "provable security." We have seen a bit of this already in our analysis of stream ciphers and block ciphers, when we used definitions of PRG and PRP distinguishing advantage.

At a high level, the provable security approach works in three distinct steps: (1) Defining security goals, (2) Defining computational assumptions, and (3) Proving security via reductions. Each of these steps involves idiosyncrasies particular to cryptography. In CS 284, we'll be looking at (1) mostly, with the idea of learning about how definitions can be used as a tool for evaluating cryptography. The other steps require a little background in complexity theory and will only be mentioned in passing (but are covered in great detail in CS 384).

### 8.2.1 Defining Security Goals

The first step is finding a definition of "security" that enumerates what an adversary is allowed to do and how much computational effort it my expend. In this step we seek to formalize something like

*"We should prevent an adversary mounting a chosen-plaintext attack, and employ any algorithmic strategy in* $2^{100}$ *time, from learning any non-trivial information about plaintexts it does not already know."*

The salient features of this informal definition include the adversary capabilities (chosen-plaintext attack), a resource bound ($2^{100}$ time), and a goal (learning information about plaintexts that it does not already know). Also central to this approach is that we want to allow *any* algorithmic strategy, including ones we don't know about ourselves. This is remarkable because the current state of theoretical computer science is not very good at reasoning about the limits of general time-bounded algorithms. This situation is very similar to that of pseudorandom generators, where we wanted to think only of time-bounded distinguishers failing, but we have no real tools for proving anything conclusive about them.

## 8.3 Chosen-Plaintext Security of Randomized Encryption Schemes

We next formalize a security goal for randomized encryption schemes. The idea is to require that an adversary cannot learn anything non-trivial about (several) encrypted messages. It should not, in particular, be able to detect if the same message was encrypted multiple times or not. We'd also like our encryption scheme to securely encrypt any type of messages selected by an arbitrary underlying protocol (e.g. HTTP), and not rely on messages avoiding any pathologies.

More interestingly is that we want our encryption to remain secure *even when an adversary can influence what we encrypt.* Here is a modern example where an adversary has this type of power: Suppose we are encrypting traffic between a front-end web server and a backend database holding user account information. Anyone on the internet can create a free account, and upon account creation, the username $u$ and password $p$ are encrypted by the webserver and sent to the database. Now suppose the link between these machines is visible to an adversary. This adversary[1] can create accounts just like anyone else! Thus it can choose $u$ and $p$ as it likes, and then observe the resulting ciphertext. Moreover, it can do this many times, perhaps thousands.

More subtle versions of this problem can arise as well. Sometimes adversaries will know we're encrypting information from a network protocol. By interfering with our connection, it may be possible for the adversary to induce us to encrypt predictable error messages.

With that motivation, we give a definition that should hold up against those types of attacks: The solution is to make a definition where the adversary gets to pick the messages, sort of like in one-time CPA, but now for many messages. Formally, this definition uses the concept of an *oracle.* You can think about oracles intuitively as "subroutines" that an algorithm can call. When $\mathcal{A}$ is an oracle algorithm and $O_1$ is a function, we write $\mathcal{A}^{O_1}$ for $\mathcal{A}$ connected to the oracle (subroutine) $O_1$. If $O_2$ is another oracle, we write $\mathcal{A}^{O_2}$ for $\mathcal{A}$ connected to $O_2$, and so on. A key point in this formalism is that $\mathcal{A}$ can only observe the input/output behavior of its oracle, and *not the code implementing the oracle.* So if $O_1$ and $O_2$ are the same function, then $\mathcal{A}^{O_1}$ and $\mathcal{A}^{O_2}$ will behave exactly the same.

Now for the definition. It uses the notation $x \xleftarrow{\$} X$ to mean "select a random sample from the set $X$, and call it $x$."

**Definition 8.2.** *Let* $\Pi = (\mathsf{Enc}, \mathsf{Dec})$ *be a randomized encryption scheme with key-space* $\mathcal{K}$*, message-space* $\mathcal{M} \subseteq \{0,1\}^*$*, randomness-space* $\mathcal{R}$*, and ciphertext-space* $\mathcal{C}$*. Let* $\mathcal{A}$ *be an algorithm. Define algorithm* $\mathbf{Expt}_{\Pi}^{\mathrm{cpa}}(\mathcal{A})$ *as*

[1] This adversary may really be multiple coordinating people or machines, but we consider it to be *one* adversary.

```
Alg $\mathbf{Expt}_\Pi^{\mathrm{cpa}}(\mathcal{A})$
01 Pick $k \xleftarrow{\$} \mathcal{K}, b \xleftarrow{\$} \{0,1\}$
02 Run $\mathcal{A}^{\mathrm{LR}_{k,b}(\cdot,\cdot)}$, where the oracle is given below. Eventually $\mathcal{A}$ halts with output $\hat{b}$
03 If $\hat{b} = b$: Output 1
04 Else: Output 0

Oracle $\mathrm{LR}_{k,b}(m_0, m_1)$
    If $m_0, m_1$ are not the same length: Return $\perp$
    Pick $r \xleftarrow{\$} \mathcal{R}$
    Compute $c \leftarrow \mathsf{Enc}(k, m_b, r)$
    Return $c$.
```

*We define the* CPA advantage of $\mathcal{A}$ against $\Pi$ *to be*

$$\mathbf{Adv}_\Pi^{\mathrm{cpa}}(\mathcal{A}) = \left|\Pr[\mathbf{Expt}_\Pi^{\mathrm{cpa}}(\mathcal{A}) = 1] - \frac{1}{2}\right|.$$

The algorithm $\mathbf{Expt}_E^{\mathrm{cpa}}(\mathcal{A})$ should be thought of as a "test-harness" for $\mathcal{A}$. This algorithm is not something we would ever use in practice, as it does not nothing useful other than evaluate a potential adversary $\mathcal{A}$. As with pseudorandom generators, the advantage $\mathbf{Adv}_E^{\mathrm{cpa}}(\mathcal{A})$ is a "score", this time between 0 and 1/2; The higher the advantage, the better the adversary.

Most of the complexity in this definition is in $\mathbf{Expt}_E^{\mathrm{cpa}}(\mathcal{A})$. The experiment picks a random key $k$ and a random bit $b$. The adversary then runs with access to an oracle $\mathrm{LR}_{k,b}(\cdot,\cdot)$ which uses $k$ and $b$. The goal of this adversary is to guess $b$; Eventually it will output its guess $\hat{b}$. When querying the oracle, the adversary is picking left/right messages $m_0, m_1$, and it is given as many shots as it likes (subject to any limitations on its runtime, since each query costs at least one basic operation). Note that each query is answered by picking a fresh, independent value $r$ to be used for encryption. Also, every call to the oracle uses the same values of $k$ and $b$. That is, the same key is used, and whether the left or right message is encrypted is determined across all queries, not per-query (so either *every* query is answered by encrypting the left message, or *every* query is answered by encrypting the right message).

Note that even a very simple $\mathcal{A}$ can win with probability 1/2, say by always outputting 1 or always 0. The goal of the adversary is to do better than that, so the definition subtracts 1/2 (if we didn't do this, we could just mentally calibrate our expectations, but this is simpler). In practice, an adversary running in time (say) $2^{100}$ and achieving $\mathbf{Adv}_E^{\mathrm{cpa}}(\mathcal{A}) = 1/2^{100}$ may be considered a "break," depending on the exact parameters like key-length.

In practice we are concerned with adversaries $\mathcal{A}$ with a similar level of resources to those we considered before. For example, we might hope for security against an adversary running time $2^{128}$ and issuing $2^{128}$ queries, achieving advantage $2^{-128}$. As before, the maximum advantage is 1/2, and anything remotely large (like $2^{-50}$) is considered to be a complete break.

The next claim shows that deterministic encryption schemes cannot have good CPA security.

**Claim 1.** *Suppose* $\Pi = (\mathsf{Enc}, \mathsf{Dec})$ *is an encryption scheme with* $\mathcal{K}, \mathcal{M}, \mathcal{R}$, *where* $\mathcal{M} \subseteq \{0,1\}^*$ *contains at least two equal-length messages. Assume* $\mathsf{Enc}$ *is deterministic (i.e. it does not use its input from* $\mathcal{R}$*). Then there exists an adversary* $\mathcal{A}$ *making two queries such that*

$$\mathbf{Adv}_\Pi^{\mathrm{cpa}}(\mathcal{A}) = 1/2.$$

*Proof.* Let $m, m' \in \mathcal{M}$ be distinct messages of the same length. The adversary $\mathcal{A}$ works as follows: Given access to an oracle $\mathcal{O}(\cdot,\cdot)$, do the following:

```
Query $\mathcal{O}(m,m)$ and call the response $c_1$
Query $\mathcal{O}(m,m')$ and call the response $c_2$
If $c_1 = c_2$ output 0, Else output 1.
```

We claim that $\Pr[\mathbf{Expt}_{\Pi}^{\mathrm{cpa}}(\mathcal{A}) = 1] = 1$. This is equivalent to saying that $\mathcal{A}$ always outputs the correct bit $\hat{b} = b$. This follows by the assumption that $\mathsf{Enc}$ is deterministic: if $b = 0$, then $c_1$ and $c_2$ are both $\mathsf{Enc}(k,m)$ and $\mathcal{A}$ outputs $\hat{b} = 0$. If $b = 1$ then $c_1 = \mathsf{Enc}(k,m)$ and $c_2 = \mathsf{Enc}(k,m')$. But then $c_1$ and $c_2$ cannot be equal, because decryption must be correct. Thus $\mathcal{A}$ will output $\hat{b} = 1$ in this case, completing the proof. □

**Exercise 8.1.** *Check the above proof and pinpoint where we used the fact that $m$ and $m'$ have the same length, and where we used the fact that they distinct. What happens if we relax either of these conditions?*

Notice how the adversary above was described: It's defined to work with *any* oracle $\mathcal{O}$ that fits the correct syntax (i.e. accepting two messages). We have to describe $\mathcal{A}$ in this way because it cannot "see" what oracle its interacting with. But when it comes time to analyzing $\mathcal{A}$, we actually instantiate $\mathcal{O}$ with a particular function and calculate probabilities.

**Exercise 8.2.** *Let $\Pi$ be the OTP cipher, viewed as an encryption scheme with deterministic encryption scheme. Describe a simple adversary $\mathcal{A}$ that never inputs the same message to its oracle more than once and yet $\mathbf{Adv}_{\Pi}^{\mathrm{cpa}}(\mathcal{A}) = 1/2$. Note the above adversary inputs $m$ more than once (in fact three times) to its oracle, so that adversary does not count.*

**Exercise 8.3.** *Define $\mathsf{Enc} : \mathcal{K} \times \mathcal{M} \times \mathcal{R} \to \mathcal{C}$, where $\mathcal{K} = \mathcal{M} = \mathcal{R} = \{0,1\}^{\ell}$ and $\mathcal{C} = \{0,1\}^{2\ell}$, as*

$$\mathsf{Enc}(k,m,r) = (r \oplus k, m \oplus r).$$

- *Give an algorithm $\mathsf{Dec}$ that makes $\Pi = (\mathsf{Enc}, \mathsf{Dec})$ into a randomized encryption scheme (i.e. so that $\mathsf{Dec}$ correctly recovers the message).*
- *Give a simple adversary $\mathcal{A}$ such that $\mathbf{Adv}_{\Pi}^{\mathrm{cpa}}(\mathcal{A}) = 1/2$.*

Those exercises only gave us some examples of encryption schemes with bad CPA security. The following exercise guides our search for good constructions by showing that we cannot achieve *perfect* CPA security against *all* adversaries. The proof is a little technical and omitted, but you may like to try proving it.

**Exercise 8.4** (Harder, optional)**.** *Let $\Pi$ be a randomized encryption scheme. Show there exists an adversary $\mathcal{A}$ (not necessarily efficient) such that $\mathbf{Adv}_{\Pi}^{\mathrm{cpa}}(\mathcal{A}) > 0$.*

Construction encryption schemes with good CPA security is usually done from block ciphers in practice, and that is the subject of the next set of notes.